L19: Entry 9 of 35 File: USPT Dec 4, 2001

DOCUMENT-IDENTIFIER: US 6327699 B1
TITLE: Whole program path profiling

claim 13 best

Detailed Description Text (24):
PPCompress 250 receives the executed path trace and outputs a whole program path. The trace may be received real time and compressed, or it may be stored and then accessed by PCCompress 250 as desired. PPCompress 250 compresses the executed path trace to produce a smaller and manageable representation that also uncovers its regular structure. This is accomplished by compressing the stream of acyclic paths, the executed path trace, into a context-free grammar as shown above, that produces exactly the program's path. The resulting grammar both reflects hierarchical structure in the input string and is typically far more compact than the original path trace. The directed acyclic graph representation of this grammar is the whole program path. The whole program path contains the program flow including crossing loop boundaries.

Detailed Description Text (26):
The SEQUITUR algorithm is used with a modification to it to compress the executed path trace into the whole program path. The modification involves looking ahead a single symbol before introducing a new rule to eliminate a duplicate digram. A pseudocode representation of the modified SEQUITUR algorithm is shown in FIG. 5. The modified algorithm takes a string as input. Each repetition in the string gives rise to a new rule in the grammar and is replaced by a non-terminal symbol. LHS is the left side or non-terminal side of a grammar production. RHS is the right side of the production. This is clearly shown in FIG. 3 at grammar 340, where the individual S, A, B, and C are on the left side, with the other symbols comprising the right side. In the pseudocode representation, two while statements are used to control the flow of the algorithm. The first loops through input characters corresponding to the trace while there are more characters to process. c is defined as the next input character and is appended to the start of a rule, S. The second loop is performed while property is violated to create new rules and compress the trace into the grammar.

CLAIMS:

4. The method of claim 1 wherein producing the whole program path from the executed path trace comprises compressing the executed path trace to produce the whole program path.

9. A system comprising:

an instrumentation tool to add instructions into a program;

an execution unit coupled to the instrumentation tool to execute the program and output an executed path trace; and

a compression unit coupled to the execution unit that receives the executed path trace and outputs a whole program path.

12. The system of claim 9, wherein the compression unit uses an on-line algorithm

on the executed path trace to produce the whole program path.

First Hit Fwd Refs

Generate Collection Print

L27: Entry 2 of 158 File: USPT Apr 13, 2004

DOCUMENT-IDENTIFIER: US 6721875 B1
TITLE: Method and apparatus for implementing a single-syllable IP-relative branch instruction and a long IP-relative branch instruction in a processor which fetches instructions in bundle form

Detailed Description Text (14):
There are five IA-64 instruction syllable types (M, I, F, B, and L), six IA-64 instruction types (M, I, A, F, B, and L), and 12 basic template types (MII, MI,I, MLX, MMI, M,MI, MFI, MMF, MIB, MBB, BBB, MMB, MFB). An instruction template type specifies the mapping of instruction syllables to execution unit types, as well as the location of "stops". Each basic template type has two versions: one with a stop after the third syllable, and one without. A stop indicates that one or more instructions after the stop may have certain kinds of resource dependencies with respect to one or more instructions before the stop (i.e., a stop enables explicit specification of parallelism). Note that the MI,I and M,MI instruction templates, by definition, have mid-bundle stops (denoted by the commas in these templates). Instructions must be placed in syllables corresponding to their instruction types, and based on a template specification, except for A-type instructions, which can be placed in either I or M syllables. For example, a template specification of MII means that of the three instructions in a bundle 100, the first instruction is a memory (M) or A-type instruction, and the next two instructions are integer (I) or A-type instructions.

Detailed Description Text (60):
Each instruction bundle pair stored in the L0I 204 maps to a single branch target (or portion thereof) stored in the L0IBR 202. A target therefore maps to the first sequentially executed branch instruction of an instruction bundle having a predicted taken branch, which instruction bundle is the first bundle following a "bundle pair entry point". Since IA-64 software architecture dictates that program flow control can be directed to the start of any instruction bundle 100, a branch instruction may redirect program flow control to either the first (BUNDLE_0) or second (BUNDLE_1) instruction bundle of a bundle pair. It is therefore assumed that when program flow control is directed (or redirected) to one bundle of a bundle pair during a current fetch of the bundle pair, program flow control will be directed to the same bundle at the next fetch of the bundle pair. The point to which program flow control is directed is referred to herein as a bundle pair entry point. Once a bundle pair entry point has been determined, the first predicted taken branch may be determined by assessing the trigger prediction bits corresponding to the instruction bundle pair and then determining which branch instruction of the instruction bundle pair is the first branch instruction after the bundle pair entry point to denote a predicted taken branch.

Detailed Description Text (96):
Branch prediction information fetched from the L0IBR 202 is also passed down a pipeline (i.e., through rotator mux 234 and a branch execution unit 236 (which may be one of many)). When a branch corresponding to the prediction information retires at the back-end of a pipeline, it is passed to the non-speculative new branch prediction structure 212 along with its current fetch prediction information, at which time a non-speculative next fetch prediction can be made for the branch. A non-speculative branch prediction is passed to multiplexer 228, and can be used to

fix-up an incorrect speculative prediction which was stored to the ISBBR 226 or L0IBR 202.

CLAIMS:

7. A processor as in claim 6, wherein each instruction bundle further comprises a template field, wherein one state of the template field maps a first syllable of an instruction bundle to a memory execution unit, and maps second and third syllables of an instruction bundle to the first branch execution unit.

Refine Search

Search Results -

Terms	Documents
L36 and computation$	188

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L37

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Tuesday, June 22, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L37	L36 and computation$	188	L37
L36	L35 and (symbol$ Or phrase$ or text$)	417	L36
L35	map$ near4 (many$ near9 one$)	708	L35
L34	map$ near4 (many near4 to near4 one$)	0	L34
L33	L31 and (architect$ near5 stat$ near8 vector$)	1	L33
L32	L31 and architect$	420	L32
L31	L30 and (recurrent$ or repet$) and (compute or computation$ or comutable$)	505	L31
L30	L29 and (phrase$ or text$)	2579	L30
L29	L28 and (processor$ or execut$) near5 instruction$	5135	L29
L28	map$ and ((many near8 one) or (n-dimension$ near4 one$))	23545	L28
L27	L26 and (many near4 one$)	158	L27
L26	map$ near9 instruction$ near9(proecessor$ or execut$)	902	L26

L25	L24 and map$ and symbol$	1	L25
L24	6412066.pn.	1	L24
L23	(map$ near9 architec$ near9 symbol$)	14	L23
L22	l20 and (reusable$ Or reuse$)	1	L22
L21	L20 and (compute or computable$)	1	L21
L20	6327699.pn.	1	L20
L19	(execut$ near4 trace$) near9 (compac$ or compres$ or reduc$ or supress$ or press$)	35	L19
L18	(compress$ near4 execut$) near9 (repeat$ or repe$ or recurrent$ or reusable or resue)	46	L18
L17	L13 and (reus$ near4 (computation$ or compute))	0	L17
L16	L13 and (reus$ near4 (computation$ or compute$))	8	L16
L15	L13 and (reus$ near4 (computation$ or comkpute$))	0	L15
L14	L13 and (reus$ near4 comput$)	8	L14
L13	(compress$ near4 execut$)	2754	L13
L12	L11 and (recurrent$ or repe$ or re$)	1	L12
L11	L9 and execut$	1	L11
L10	L9 and execut$ near9 (trace$ or profile$ or performanc$)	0	L10
L9	5937194.pn.	1	L9
L8	L7 and (compress$ or press$ or reduc$ or supress$ or cmpact$) near9 (execut$ or trace$)	12	L8
L7	(reuse or reusable$) near9 (compute or computation$)	106	L7
L6	L5 and execut$ near5 (trace$ or tracing$)	1	L6
L5	(reus$ near5 comput$) and (compress$ or supress$) and (recurrent$ or replicat$ or repe$) and execut$	66	L5
L4	L3 and execut$ near5 (trace$ or tracing$)	1	L4
L3	(reus$ near5 comput$) and (compress$ or supress$) and (recurrent$ or replicat$ or repeat$) and execut$	65	L3
L2	(reus$ near5 comput$) and (compress$ or supress$) and (recurrent$ or replicat$) and execut$	10	L2
L1	(state$ near4 vector$) near5 ((instanc$ OR OBJECT$)near4 (instruction$ or process$))	7	L1

END OF SEARCH HISTORY

L19: Entry 1 of 35 File: USPT Feb 17, 2004

DOCUMENT-IDENTIFIER: US 6694488 B1
TITLE: System for the design of high-performance communication architecture for system-on-chips using communication architecture tuners

Detailed Description Text (34):
An embodiment of the overall algorithm for designing CAT-based communication architectures is shown in FIG. 10. In step 1, performance analysis is performed on the partitioned/mapped system description in order to derive the information and statistics used in the later steps. In the present work, the performance analysis technique presented in Lahiri et al, which is comparable in accuracy to complete system simulation, while being much more efficient to employ in an iterative manner. For more details, see K. Lahiri, A. Raghunathan and S. Dey, "Fast Performance Analysis of Bus Based System-on-Chip Communication Architectures," in Proc. Int. Conf. Computer-Aided Design, Nov. 1999. The output of this analysis is a communication analysis graph, (CAG) which is a highly compact representation of the system's execution under the given input traces. The vertices in the graph represent clusters of computations and abstract communications performed by the various components during the system execution. The edges in the graph represent the inter-dependencies between the various computations and communications. Note that since the communication analysis graph is effectively unrolled in time, it is cyclic, and may contain several distinct instances of a single computation operation or communication from the system specification. The communication analysis graph is constructed by extracting necessary and sufficient information from a detailed system execution trace. See K. Lahiri, A. Raghunathan and S. Dey, ".degree. Fast Performance Analysis of Bus Based System-on-Chip Communication Architectures," in Proc. Int. Conf. Computer-Aided Design, Nov. 1999. The CAG can be easily analyzed to determine various performance statistics such as system critical path, average processing time, number of missed deadlines, etc.

For claim 13.

Refer previous Non-pat ref and this.

L23: Entry 4 of 14 File: USPT Jun 25, 2002

DOCUMENT-IDENTIFIER: US 6412066 B1
TITLE: Microprocessor employing branch instruction to set compression mode

Detailed Description Text (41):
Additionally, the instruction tables use several symbols. In an operands column 102, the symbols rs, rt, xs, xt, and rd are used. Rs and xs refer to second register field 26 (or second register field 76), while rt and xt refer to first register field 24 (or first register field 74). Similarly, rd refers to third instruction field 32 (or third instruction field 82). As mentioned for one embodiment above, first register field 24, second register field 26, and third register field 32 comprise three bits each. Table 1 below lists the mapping of the field encodings (listed in binary) to registers in the MIPS RISC architecture for these symbols. Other mappings are also contemplated, as shown further below. Names assigned according to MIPS assembler convention are also listed in Table 1.

mapping architectural Stat. into symbol

First Hit Fwd Refs

End of Result Set

Generate Collection Print

L23: Entry 14 of 14 File: USPT Aug 16, 1994

DOCUMENT-IDENTIFIER: US 5339419 A
TITLE: ANDF compiler using the HPcode-plus compiler intermediate language

CLAIMS:

20. The computer software compiler system of claim 19, wherein said low-level code generator comprises:

(1) means for mapping said variables to the memory architecture of the target computer program by referring to the machine configuration file and to said symbol table;

(2) means for mapping said data types to the memory architecture of the target computer program by referring to the machine configuration file and to said type table.

55. The computer software compiler method of claim 54, wherein said low-level code generator step comprises the steps of:

(a) mapping said variables to the memory architecture of the target computer program by referring to the machine configuration file and to said symbol table;

(b) mapping said data types to the memory architecture of the target computer program by referring to the machine configuration file and to said type table.

Refine Search

Search Results -

Terms	Documents
(compress$ near4 execut$) near9 (repeat$ or repe$ or recurrent$ or reusable or resue)	46

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L18

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Tuesday, June 22, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L18	(compress$ near4 execut$) near9 (repeat$ or repe$ or recurrent$ or reusable or resue)	46	L18
L17	L13 and (reus$ near4 (computation$ or compute))	0	L17
L16	L13 and (reus$ near4 (computation$ or compute$))	8	L16
L15	L13 and (reus$ near4 (computation$ or comkpute$))	0	L15
L14	L13 and (reus$ near4 comput$)	8	L14
L13	(compress$ near4 execut$)	2754	L13
L12	L11 and (recurrent$ or repe$ or re$)	1	L12
L11	L9 and execut$	1	L11
L10	L9 and execut$ near9 (trace$ or profile$ or performanc$)	0	L10
L9	5937194.pn.	1	L9
L8	L7 and (compress$ or press$ or reduc$ or supress$ or cmpact$) near9 (execut$ or trace$)	12	L8

L7	(reuse or reusable$) near9 (compute or computation$)	106	L7
L6	L5 and execut$ near5 (trace$ or tracing$)	1	L6
L5	(reus$ near5 comput$) and (compress$ or supress$) and (recurrent$ or replicat$ or repe$) and execut$	66	L5
L4	L3 and execut$ near5 (trace$ or tracing$)	1	L4
L3	(reus$ near5 comput$) and (compress$ or supress$) and (recurrent$ or replicat$ or repeat$) and execut$	65	L3
L2	(reus$ near5 comput$) and (compress$ or supress$) and (recurrent$ or replicat$) and execut$	10	L2
L1	(state$ near4 vector$) near5 ((instanc$ OR OBJECT$)near4 (instruction$ or process$))	7	L1

END OF SEARCH HISTORY